Virtual Dispatch Alternatives to Vtables

25 September, 2001

George Bosworth
CLR Software Architect

Agenda

- Observations and Biases
- Current Approach (V1) - vtables
- Alternative - stubs and heuristics
- Implementation
- Performance Measurements
- Observations

Observations and Biases

- Delayed binding is in general a good thing
  - loose coupling, resilience, reuse, …
- Vtables are a bad thing in the large
  - space inefficient, tight bindings, …
- Heuristics can mask a lot of ignorance and provide for the unexpected

Agenda

- Observations and Biases
- Current Approach (V1) - vtables
- Alternative - stubs and heuristics
- Implementation
- Performance Measurements
- Observations

Current Approach (V1)

- Method table contain
  - vtables for virtual functions
  - vtables for interface functions
- Interface Table maps interfaces for a type to the interface vtable in a method table

Method table

```
interface I1 {m1(); }
interface I2 {m2(); m3(); }
class A : I1 {m1(){..}; m2(){..}; }
```

A.m1, I1.m1
A.m2

```
class B : A, I2 {m3(){..}; }
```

A.m1, I1.m1
A.m2; I2.m2
B.m3, I2.m3

```
class C : B { I2.m2() {..}; }
```

A.m1, I1.m1
A.m2
B.m3
I2.m2
I2.m3

Method table

```
interface I1 {m1(); }
interface I2 {m2(); m3(); }
class A : I1 {m1(){..}; m2(){..}; }
```

A.m1, I1.m1
A.m2

```
class B : A, I2 {m3(){..}; }
```

A.m1, I1.m1
A.m2, I2.m2
B.m3, I2.m3

```
class C : B { I2.m2() {..}; }
```

A.m1, I1.m1
A.m2
B.m3
I2.m2
I2.m3

Calling Sequences

- virtual call: c.m2()

```
;virtual invocation of m2(), ecx = this ptr
mov  eax,[ecx]     ;get method table
call [eax+4]       ;dispatch thru virtual slot 1
```

- interface call: i2.m2()

```
;interface invocation of I2.m2(), ecx = this ptr
mov  eax,[ecx]     ;get method table
mov  eax,[eax+xx] ;get the interface map for class C
mov  eax,[eax+yy]  ;get the dispatch vtable of I2 in C
                   ;  which is virtual slot 3 in C
call [eax+0]       ;dispatch thru slot 0 of I2 in C
```

Interface Map

- Tightly packed two dimensional array
  - each class points to its row
  - interface ids are monotonically increasing global index

Method Table Statistics

	Size in bytes						
program	# MT	MT	VT	Static	Field	GC	I/Fmap
helloworld.exe	269	53076	21788	14352	636	1688	2776
vswinformsdesigner.	2072	464008	236440	60860	8716	21144	25680
vsstartuppropertygr	761	185328	91512	39688	3684	6072	10888
winformsstarupcomp	705	159728	72340	39008	3604	5748	8008
xsltest.exe	647	115400	48612	25260	2688	5228	5144
oledbtest.exe	987	162044	77616	40840	3764	7860	8536

Winforms Hierarchical Slice

	Size in bytes						
class	MT	[illegible]	Static	Field	GC	IFmap	parent
Object	100	16	40	0	0	0	
MarshalByRefObject	124	28	40	0	12	8	System.Object [mscorlib.dl
Component	156	52	20	4	20	16	System.MarshalByRefOb
Control	3664	1268	[illegible]	416	28	136	System.ComponentMode
ScrollableControl	1616	1280	116	4	36	136	System.Windows.Forms.C
ContainerControl	1640	1316	88	4	44	144	System.Windows.Forms.S
Form	2584	1380	216	248	52	144	System.Windows.Forms.C
[illegible]	1940	1388	240	64	60	144	System.Windows.Forms.F

Type and Slot Usage

[illegible]

threshold	[illegible]	#used	[illegible]	#used	[illegible]	[illegible]
1	164	8	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	164	4	[illegible]	11	[illegible]	11
4	164	1	[illegible]	[illegible]	[illegible]	[illegible]
8	164	1	[illegible]	1	[illegible]	8

A bigger program

viewinformsdesigner

threshold	#M	#used	#slots	#used#	M-slots	#calls
1	[illegible]	440	57519	[illegible]	[illegible]	[illegible]
[illegible]	1067	414	57519	2874	21664	2173538
4	[illegible]	369	[illegible]	14	[illegible]	[illegible]
8	[illegible]	148	[illegible]	[illegible]	14005	[illegible]
16	[illegible]	[illegible]	[illegible]	1341	[illegible]	[illegible]
[illegible]	1067	[illegible]	57519	940	17186	215907
64	[illegible]	[illegible]	[illegible]	832	15805	2148738
128	[illegible]	180	[illegible]	[illegible]	[illegible]	[illegible]
256	[illegible]	144	[illegible]	455	8514	[illegible]
[illegible]	[illegible]	108	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
2048	1067	44	57519	[illegible]	[illegible]	839087
[illegible]	1067	36	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	41	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	1067	11	57519	[illegible]	439	[illegible]

Observations

- Most invocations go thru few slots
- Not many types actually instantiated
- Something weird going on in Winforms
  - 11 slots account for ½ the calls
  - 4 of those slots are in HashTable

Other programs

Limitations

- vtables require the code generator (jit) to bind to a slot number
  - cannot decide later (dynamic invocation for scripting)
  - use another binding approach (multimethods)
  - detect out of range (version resiliency/ extensibility)
- Must be built whether needed or not

What is needed

- Mapping function from:
  - (calling frame, ref token)
- to:
  - method implementation address
- Vtables are a restricted form of this mapping function where:

Asymmetries to leverage

- Few actual types instantiated
- Small number of slots used

- Assumption we will make:
  - most call sites are mostly monomorphic

Alternative Stub Dispatch

- Type and method specific stubs
- Different kinds of stubs
  - mostly monomorphic
  - polymorphic
- Stubs built on demand as needed
- Heuristics used to choose and adjust the kind of stub in use

Simple Logical Stub Model

- generated call sites:
  - aligning pad/nop if necessary (0-3 bytes)
  - call stubaddress
- initial stubaddress points to:
  - push [illegible]
  - [illegible]
- after use, stubaddress points to:
  - [illegible]
  - jne fail
  - [illegible]
- [illegible]

Trace based Measurements

- Traces from a modified fjit
- Stub model consumed traces

Trace for OLEDBTEST

(Cont) Trace for OLEDBTEST

Summary for OLEDBTEST

jitted sites	3273	called sites	2130	
stub calls	864968	misses	24457	(2.8%)
stubs made	897	tokens	121	
stubs/site	[illegible].17	stub refs	5.54	

- small number of stubs and tokens
- low miss rate
- program has
  - 987 types
  - over 10,000 methods
  - 20,000 vtable slots

Mono or Polymorphic ?

Modified CLR built

- EE generates and manages stubs
- Jit generates stub calls for interface and virutal methods
- What wasn't done:
  - eliminate the vtables themselves
  - integrate stubs with app domain unloading
  - atomic update of call sites partially undone

Tokens

- tokens composed of two 32bit fields
  - contract
    - 0 means use the type of "this"
    - >0 means contract is the interface id
  - member
    - slot number in the vtable or interface table
- Since the numbers are small, it is usually (always) bit encoded in 31 bits

Call Sites

- with call site rewriting

```
mov eax,[ecx]     //preload the MT
call stub         //direct call to stub
```

- as pure code

```
mov eax,[ecx]     //preload the MT
call [stubcell]   //indirect call to stub
```

Lookup Stubs

- Call sites initially point to:

```
push token
jmp  ResolveWorkerStub
```

- ResolveWorkerStub enters the EE and causes a Dispatcher stub to be found (or created) that is specific to the type of the "this" pointer of the call
  - betting monomorphic until shown otherwise

Dispatcher Stubs

```
cmp eax,expectedMT
jne failstub      ;must be fwd branch
jmp targetaddress
```

- Checks that the actual type of «this» is what is expected, and if so transfers to the method implementation
- Fail stub will cause a lookup to be done and then pass control to the correct method.
- One per tuple (expectedMT, token)

Fail Stub

```
sub  [counter],1
jl   patchstub  ;must be fwd jmp
;fall into resolve stub
```

- Fail stub checks to see if we are failing a lot; counter is periodically inc'd
- Patch stub will patch the call site to point to the resolve stub
- Resolve stub will do a cache lookup
- One per token; i.e., dispatch stub share

Patch Stub

call PatchWorkerStub
jmp resolvestub

- PatchWorkerStub goes into the EE and decides whether or not the call site should be patched to point to the resolvestub directly, i.e. make it a polymorphic call site.

Resolve Stub

Resolve Stub

```
[illegible]
mov edx,[ecx]
[illegible]
```

Resolve Stub (Cont)

- Resolve stubs to a cache lookup keyed by the type of <this> and the token.
- Cache contains the method impl address.
- Key to the cache hit rate is the hash fn used
  - Current has not been extensively tuned

Agenda

- Observations and Biases
- Current Approach (V1) – vtables
- Alternative – stubs and heuristics
- Implementation
- Performance Measurement
- Observations

Simple calls - best case

[illegible]

Loop is just the loop

	loop count	[illegible]	[illegible]	[illegible]
Loop	100000	0	0	[illegible]
Loop	1000000	0	[illegible]	[illegible]
Loop	10000000	[illegible]	[illegible]	[illegible]
Call	100000	[illegible]	[illegible]	[illegible]
Call	1000000	[illegible]	[illegible]	[illegible]
Call	10000000	[illegible]	[illegible]	[illegible]
Static	100000	[illegible]	[illegible]	[illegible]
Static	1000000	[illegible]	[illegible]	[illegible]
Static	10000000	[illegible]	[illegible]	[illegible]
Interface	100000	[illegible]	[illegible]	[illegible]
Interface	1000000	[illegible]	[illegible]	[illegible]
Interface	10000000	[illegible]	[illegible]	[illegible]
Virtual	100000	[illegible]	[illegible]	[illegible]
Virtual	1000000	[illegible]	[illegible]	[illegible]
Virtual	[illegible]	[illegible]	[illegible]	[illegible]

Richards' worse case

- Highly polymorphic
  - [illegible]
  - [illegible]
  - [illegible]

	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
dispatcher stubs	[illegible]	[illegible]	[illegible]
dispatcher [illegible]	[illegible]	[illegible]	[illegible]
dispatcher [illegible]	[illegible]	[illegible]	[illegible]
resolver stubs	[illegible]	[illegible]	[illegible]
resolver [illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]

The bad news

- The timings are very unstable. Small changes in unrelated portions of the EE can cause big jumps (factor of 2) in the above numbers.
- Tried, unsuccessfully, to isolate via vtune.
- Built a vtune post processor that computed aggregate stats for the stubs themselves, still didn't isolate

Controlled Sequences

- Built a hardcoded set of code sequences that did the current virtual dispatch and the stub dispatch
- Varied the calling and the called environment
- Basically a loop of 20,000 around 50 inlined call sites

Controlled Sequences (Cont)

- NOP(s) of 1 or 2 bytes are inserted between every 10 calls
- (CONTEXT 0) = no inter call instr overhead
- (CONTEXT 5) = 5 instr (dummy bp frame-like) inter call overhead
- Called routine was just a return, and was either Frameless or EBP Framed

Controlled Sequences (Cont)

	[illegible]		[illegible]	
	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

- anomalies make one read the intel perf manuals really close

does it vary with code size?

MA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
MA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
MA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
MA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
MA	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

- small - loop 20,000 over 50 inlined sites
- large - loop 1,000 over 1,000 inlined sites

Say hello to the BTB

- Branch Transfer Buffer on the Pentium
  - all my runs were on a 933 Mhz PIII with 800Mhz RDRAM

Branch basics on pentiums

- The BTB is check on every instr fetch of any kind. If it hits, it turns all transfers into zero or near zero cost
- If no BTB hit, fall thru is free
- If non-fall thru but staticly predicted, e.g. a jmp relative, cost is a half dozen or so cycles
- If not statically predicted, e.g. an indirected call or a mis-prediction cost is a score or more cycles and this doesn't include any cache or memory fetches at all
- BTB [illegible] cells [illegible] small

Simple Calls – revisited

[illegible]

- Indirections, like vtable dispatch, fall apart in the large

Agenda

- Observations and Biases
- Current Approach (V1) – vtables
- Alternative – stubs and heuristics
- Implementation
- Performance Measurement
- Observations

Conclusion

- Stubs can be performant
- Perf is hard to measure
- Things like BTB make tuning almost impossible
- Transfers of control carry a large hidden perf cost on Pentiums, since any transfer will put pressure on the BTB

Continuations, anyone?

Nick Benton
MSR Cambridge

What's a continuation?

- The "rest of the computation"
- Having first class continuations in a language or VM means being able to *reify* the current "rest of the computation" into a value which can then be passed around and invoked later
  - Possibly never
  - Possibly several times
- To a first approximation, grab the stack and turn it into an object on the heap
- This single facility allows one to build lots of fancy control structures which would normally be implemented separately

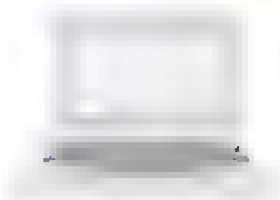
For example

- Exceptions
- Backtracking a la Prolog
- Coroutines
- Lightweight threads (CML)
- Stateless servers
- Mobile computations (migratory apps)
- Security features
- Time-travel debugging
- Arbitrary monads – adding step-counting instrumentation, non-determinism, probabilistic computation

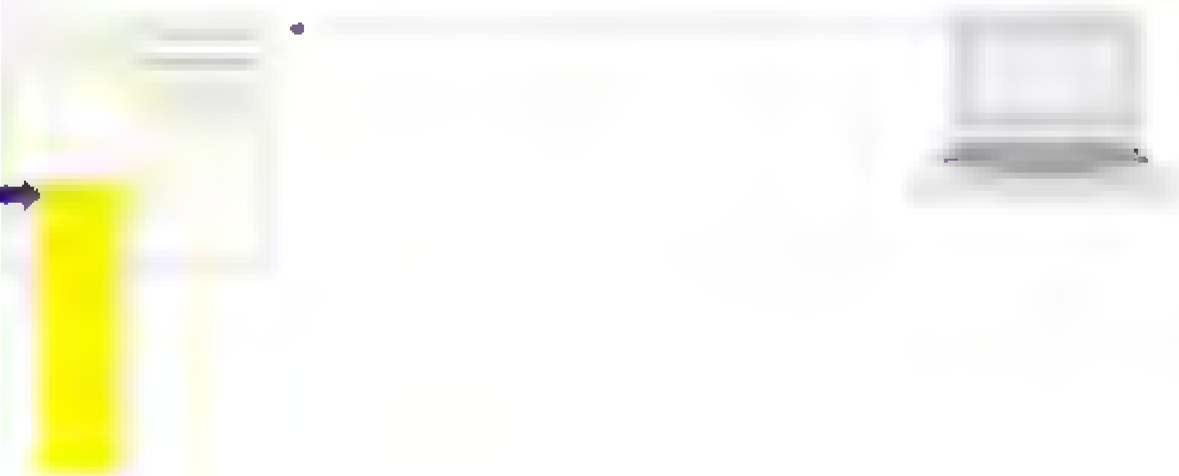
Monadic reflection using call/cc

- Add nondeterminism (for example) to already-compiled code

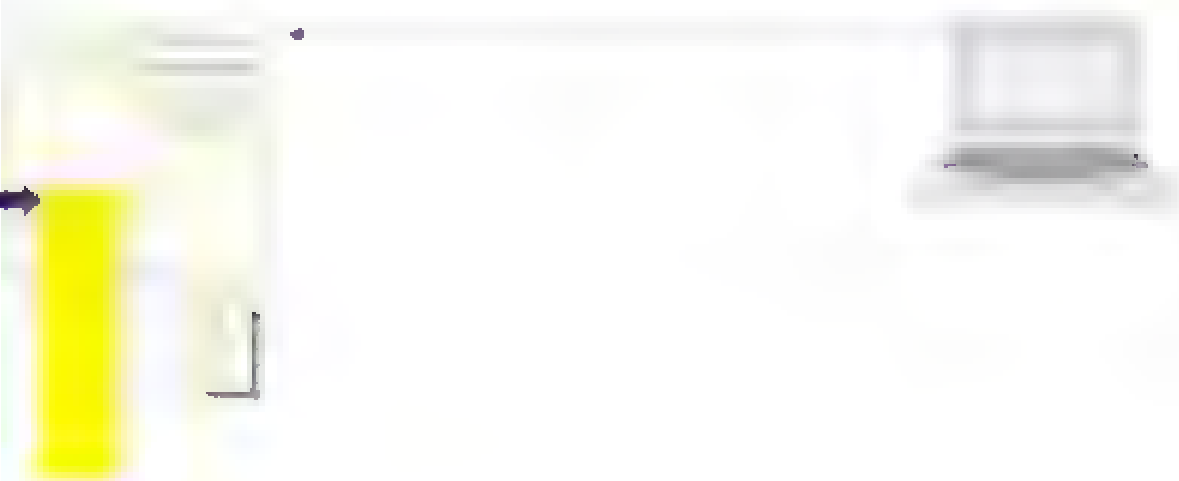
e.g. stateless servers



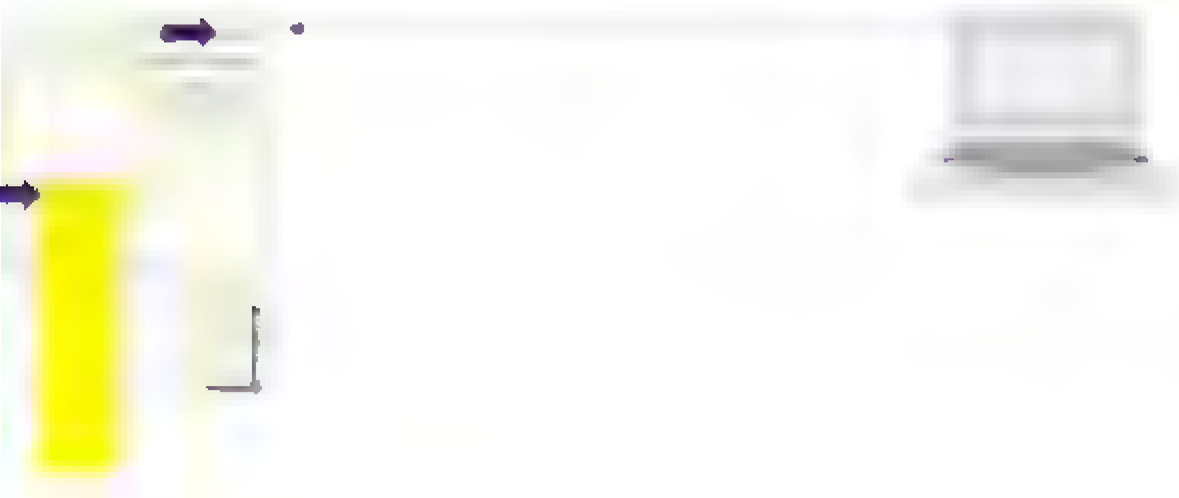
e.g. stateless servers



e.g. stateless servers



e.g. stateless servers



History

- First invented in denotational semantics in order to give a mathematical account of the meaning of existing control operators, such as jumps
- Then added to programming languages such as Scheme and SML/NJ (call/cc)
- Still interesting research going on
  - More refined control operators (prompt/reset hierarchies …)
  - Connection with classical logic

Continuation Passing Style

- There's a uniform translation of CBV (or, indeed, CBN) into CPS
  - Every function gets an extra argument "what to do with the result
  - Every call becomes a tail call

Why should a VM support first-class continuations?

- Some languages have them and compiling via explicit CPS to the runtime gives poor performance
- Similarly, some other languages have features (e.g. backtracking, large numbers of very lightweight threads) that can be implemented more efficiently with continuations
- It's potentially simpler than adding lots of control-related stuff in an ad hoc way
- A general purpose runtime should provide general purpose mechanisms

Why should a VM support first-class continuations?

- Some languages have them and compiling via explicit CPS to the runtime gives poor performance
- Similarly, some other languages have features (e.g. backtracking, large numbers of very lightweight threads) that can be implemented more efficiently with continuations
- It's potentially simpler than adding lots of control-related stuff in an ad hoc way
- A general purpose runtime should provide general purpose mechanisms

Why *shouldn't* a VM support first class continuations?

- Scary
- Almost certainly interact badly with existing features (e.g. security)
- Rule out some optimisations which might mean paying an efficiency cost in the common case of languages which don't use them

Conclusion

- None, really
- But it's something that seems worth thinking about

LZ77 data compression

- Before:
- After: <-5,8>.
- Encoding:
- Recast as "instructions" to a decompressor:

The echo instruction

- Example:

```
ldc.i4 12
add
ldind.i4
…
echo 30,3
```

- Like `#include` but on-the-fly
- Like `JSR` but caller delimits entry/exit without overhead polluting the stream

Summary

- Cuts LCC byte-code by ~40% == 0.60x
- Echo suits interpreters:
  - Clients that can't JIT or even decompress
  - e.g., "feature" code in cell phones
- Related tools:
  - Automatic generation of compressed bytecodes
  - High-tech code and XML compression
    (esp. for thin or costly pipes, service packs, ROM)
  - Simple, table-driven code generators and JITs


Dynamic Heuristically Driven Speculative Optimizations
George Bosworth